UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/708,722	11/09/2000	Stephan J. Jourdan	2207/9800	2194

25693 7590 06/12/2008

KENYON & KENYON LLP
RIVERPARK TOWERS, SUITE 600
333 W. SAN CARLOS ST.
SAN JOSE, CA 95110

EXAMINER	
LI, AIMEE J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
06/12/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	09/708,722	JOURDAN ET AL.	
	Examiner	Art Unit	
	AIMEE J. LI	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *03 March 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-19* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-19* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *24 February 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-19 have been considered.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Amendment as filed 03 March 2008.

Claim Rejections - 35 USC § 102

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

4. Claims 1-5 and 8-14 are rejected under 35 U.S.C. 102(e) as being taught by Kyker et al., U.S. Patent Number 6,578,138 (herein referred to as Kyker).

5. Referring to claim 1, Kyker has taught a cache comprising:

a. A cache line to store an instruction segment further comprising a plurality of instructions stored in sequential positions of cache line in reverse program order (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for

example, over two trace lines long…"; Figure 1; and Figure 2 – In regards to Kyker, as shown in Figure 2 and explained the in the cited lines, the loop has three main instructions and, when the backwards branch is encountered, the program jumps backwards, i.e. reverses directions, to the beginning of the loop instead of moving forward in the program. As shown in Figure 2, when L_H is encountered, previously stored instructions L_2 and L_3 are stored again in consecutive lines in the cache, e.g. the trace cache reverses the program direction and stores previously stored instructions again in the trace cache when a loop is encountered.).

6. Referring to claim 2, Kyker has taught the cache of claim 1, wherein the instruction segment is an extended block (Kyker column 1, lines 30-45 "…the target address, the backward taken branch, and any micro-ops between the two form a loop…"; column 2, line 59 to column 3, line 33 "…Exemplary trace T1 includes a total of four micro-ops…The second exemplary trace T2 includes the same loop, L_H, L_1, L_2, but does not include any micro-op preceding the loop itself…"; Figure 1; and Figure 2 – In regards to Kyker, the loop contains a plurality of instructions, i.e. block of instructions, with four instructions and includes the extra micro-op preceding the loop itself.).

7. Referring to claim 3, Kyker has taught the cache of claim 1, wherein the instruction segment is a trace (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop

until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2).

8. Referring to claim 4, Kyker has taught the cache of claim 1, wherein the instruction segment is a basic block (Kyker column 1, lines 30-45 "…the target address, the backward taken branch, and any micro-ops between the two form a loop…"; column 2, line 59 to column 3, line 33 "…Exemplary trace T1 includes a total of four micro-ops…The second exemplary trace T2 includes the same loop, L_H, L_1, L_2, but does not include any micro-op preceding the loop itself…"; Figure 1; and Figure 2 – In regards to Kyker, the loop contains a plurality of instructions, i.e. block of instructions, with three instructions and does not include the extra micro-op preceding the loop itself.).

9. Referring to claim 5, Kyker has taught a segment cache for a front-end system in a processor (Kyker column 5, line 45 to column 6, line 2 "…The trace cache also includes, for example, a control bloc **109** and a instruction decoder **111**..."; Figure 10; and Figure 11 – In regards to Kyker, the trace cache stores portions, e.g. segments, of the program and, as shown in Figures 10 and 11, is part of the system associated with decoding instructions, which is prior to execution and retirement of instructions, i.e. the back-end system.), comprising a plurality of cache entries to store instructions of instruction segments in reverse program order (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for

example, over two trace lines long…"; Figure 1; and Figure 2 – In regards to Kyker, as shown in Figure 2 and explained the in the cited lines, the loop has three main instructions and, when the backwards branch is encountered, the program jumps backwards, i.e. reverses directions, to the beginning of the loop instead of moving forward in the program. As shown in Figure 2, when L_H is encountered, previously stored instructions L_2 and L_3 are stored again in consecutive lines in the cache, e.g. the trace cache reverses the program direction and stores previously stored instructions again in the trace cache when a loop is encountered.).

10. Referring to claim 8, Kyker has taught a method comprising:

    a. Building an instruction segment based on program flow (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2), and

    b. Storing instructions of the instruction segment in a cache entry in reverse program order (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In

other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2 – In regards to Kyker, as shown in Figure 2 and explained the in the cited lines, the loop has three main instructions and, when the backwards branch is encountered, the program jumps backwards, i.e. reverses directions, to the beginning of the loop instead of moving forward in the program. As shown in Figure 2, when L_H is encountered, previously stored instructions L_2 and L_3 are stored again in consecutive lines in the cache, e.g. the trace cache reverses the program direction and stores previously stored instructions again in the trace cache when a loop is encountered.).

11. Referring to claim 9, Kyker has taught the method of claim 8, further comprising:

a. Building a second instruction segment based on program flow, and if the first and second instruction segments, overlap, extending the first instruction segment to include non-overlapping instructions from the second instruction segment (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2 – In regards to Kyker, as shown in Figure 1, the trace is built as normal, e.g. a first instruction

segment is built normally, when there is no backward-taken branch. When a backward branch is taken to form another, second instruction segment, another iteration of the loop is added to the trace, thereby extending the first segment to include the first segment.).

12. Referring to claim 10, Kyker has taught the method of claim 9, wherein the extending comprises storing the non-overlapping instructions in the cache in reverse program order in successive cache positions adjacent to the instructions from the first instruction segment (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2 – In regards to Kyker, as shown in Figure 2 and explained the in the cited lines, the loop has three main instructions and, when the backwards branch is encountered, the program jumps backwards, i.e. reverses directions, to the beginning of the loop instead of moving forward in the program. As shown in Figure 2, when L_H is encountered, previously stored instructions L_2 and L_3 are stored again in consecutive lines in the cache, e.g. the trace cache reverses the program direction and stores previously stored instructions again in the trace cache when a loop is encountered.).

13. Referring to claim 11, Kyker has taught the method of claim 8, wherein the instruction segment is an extended block (Kyker column 1, lines 30-45 "…the target address, the backward taken branch, and any micro-ops between the two form a loop…"; column 2, line 59 to column

3, line 33 "…Exemplary trace T1 includes a total of four micro-ops…The second exemplary trace T2 includes the same loop, L_H, L_1, L_2, but does not include any micro-op preceding the loop itself…"; Figure 1; and Figure 2 – In regards to Kyker, the loop contains a plurality of instructions, i.e. block of instructions, with four instructions and includes the extra micro-op preceding the loop itself.).

14. Referring to claim 12, Kyker has taught the method of claim 8, wherein the instruction segment is a trace (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2).

15. Referring to claim 13, Kyker has taught the method of claim 8, wherein the instruction segment is a basic block (Kyker column 1, lines 30-45 "…the target address, the backward taken branch, and any micro-ops between the two form a loop…"; column 2, line 59 to column 3, line 33 "…Exemplary trace T1 includes a total of four micro-ops…The second exemplary trace T2 includes the same loop, L_H, L_1, L_2, but does not include any micro-op preceding the loop itself…"; Figure 1; and Figure 2 – In regards to Kyker, the loop contains a plurality of instructions, i.e. block of instructions, with three instructions and does not include the extra micro-op preceding the loop itself.).

16. Referring to claim 14, Kyker has taught a processing engine, comprising:

a. A front end stage to build and store instruction segments (Kyker column 5, line 45 to column 6, line 2 "…The trace cache also includes, for example, a control bloc **109** and a instruction decoder **111**..."; Figure 10; and Figure 11 – In regards to Kyker, the trace cache stores portions, e.g. segments, of the program and, as shown in Figures 10 and 11, is part of the system associated with decoding instructions, which is prior to execution and retirement of instructions, i.e. the back-end system.), instructions provided therein in reverse program order (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2 – In regards to Kyker, as shown in Figure 2 and explained the in the cited lines, the loop has three main instructions and, when the backwards branch is encountered, the program jumps backwards, i.e. reverses directions, to the beginning of the loop instead of moving forward in the program. As shown in Figure 2, when L_H is encountered, previously stored instructions L_2 and L_3 are stored again in consecutive lines in the cache, e.g. the trace cache reverses the program direction and stores previously stored instructions again in the trace cache when a loop is encountered.), and

b. An execution unit in communication with the front end stage (Kyker column 5, line 61 to column 6, line 2 "…the computer system includes a processor **121**, a trace cache **101**, and a computer memory **123**…" and Figure 11).

Claim Rejections - 35 USC § 103

17. Claims 6-7 and 15-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kyker et al., U.S. Patent Number 6,578,138 (herein referred to as Kyker) as applied to claims 5 and 14 above, and further in view of Rotenberg et al.'s "A Trace Cache Microarchitecture and Evaluation" IEEE ©1999 (herein referred to as Rotenberg).

18. Referring to claim 6, Kyker has taught apparatus comprising;

a. An instruction cache system (Kyker column 5, lines 45-60 "…an exemplary embodiment of a trace cache…" and Figure 10),

b. An instruction segment system (Kyker column 5, line 45 to column 6, line 2 "…The trace cache also includes, for example, a control block **109** and a instruction decoder **111**..."; Figure 10; and Figure 11 – In regards to Kyker, the trace cache stores portions, e.g. segments, of the program.), comprising:

i. A fill unit provided in communication with the instruction cache system (Kyker column 5, lines 45-60 "…the control block **109** facilitates the build process by providing write enable commands to the data array…" and Figure 10),

ii. The segment cache of claim 5 included therein (See the above rejection of claim 5), and

19. Kyker has not taught a selector coupled to an output of the instruction cache system and to an output of the segment cache. Rotenberg has taught a selector coupled to an output of the instruction cache system and to an output of the segment cache (Rotenberg Section 2.1 Trace-Level Sequencing "…The output of the trace cache is one or more traces…"; Section 2.2 Instruction-Level Sequencing "The *outstanding trace buffers* in Fig. 2 are used to 1) construct new traces that are not in the trace cache and 2) track branch outcomes…"; and Figure 2 – In regards to Rotenberg, Figure 2 shows the output of the trace cache and the outstanding trace buffers are connected to a line that has two inputs and one output, which is a selector. The selector chooses between and existing trace in the trace cache, i.e. a trace cache hit, or a newly formed trace, i.e. a trace cache miss or misprediction.) A person of ordinary skill in the art at the time the invention was made, and as taught by Rotenberg, would have recognized that the trace cache system provides fast trace-level sequencing while providing a method to create nonexistent traces or repair mispredicted traces (Rotenberg Section 2.1 Trace-Level Sequencing "…The trace predictor and trace cache together provide fast trace-level sequencing…Instruction-level sequencing, discussed in the next section, is required to construct nonexistent traces or repair trace mispredictions."). Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the trace caches, buffers, and selector of Rotenberg in the device of Kyker to increase the speed of trace-level sequencing while using instruction-level sequencing to create new or correct mispredicted traces.

20. Referring to claim 7, Kyker has not taught an apparatus of claim 6, wherein the instruction segment system further comprises a segment predictor provided in communication with the segment cache. Rotenberg has taught an apparatus of claim 6, wherein the instruction

segment system further comprises a segment predictor provided in communication with the segment cache (Rotenberg Section 2.1 Trace-Level Sequencing "…A *next trace predictor*[14] treats traces as basic units and explicitly predicts sequences of traces…" and Figures 2). A person of ordinary skill in the art at the time the invention was made, and as taught by Rotenberg, would have recognized that the trace predictor achieves high branch prediction throughput with a single prediction per cycle (Rotenberg Section 2.1 Trace-Level Sequencing "…high branch prediction throughput is implicitly achieved with only a single trace prediction per cycle…"). Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the trace predictor of Rotenberg in the device of Kyker to achieve high branch prediction throughput in a single prediction cycle.

21. Referring to claim 15, Kyker has taught the processing engine of claim 14, wherein the front-end stage comprises:

a. An instruction cache system (Kyker column 5, lines 45-60 "…an exemplary embodiment of a trace cache…" and Figure 10),

b. An instruction segment system (Kyker column 5, line 45 to column 6, line 2 "…The trace cache also includes, for example, a control bloc **109** and a instruction decoder **111**..."; Figure 10; and Figure 11 – In regards to Kyker, the trace cache stores portions, e.g. segments, of the program.), comprising:

i. A fill unit provided in communication with the instruction cache system (Kyker column 5, lines 45-60 "…the control block **109** facilitates the build process by providing write enable commands to the data array…" and Figure 10),

ii. A segment cache (Kyker column 5, line 45 to column 6, line 2 "…The trace cache also includes, for example, a control bloc **109** and a instruction decoder **111**..."; Figure 10; and Figure 11 – In regards to Kyker, the trace cache stores portions, e.g. segments, of the program.), and

22. Kyker has not taught a. selector coupled to an output of the instruction cache system and to an output of the segment cache. Rotenberg has taught a selector coupled to an output of the instruction cache system and to an output of the segment cache (Rotenberg Section 2.1 Trace-Level Sequencing "…The output of the trace cache is one or more traces…"; Section 2.2 Instruction-Level Sequencing "The *outstanding trace buffers* in Fig. 2 are used to 1) construct new traces that are not in the trace cache and 2) track branch outcomes…"; and Figure 2 – In regards to Rotenberg, Figure 2 shows the output of the trace cache and the outstanding trace buffers are connected to a line that has two inputs and one output, which is a selector. The selector chooses between and existing trace in the trace cache, i.e. a trace cache hit, or a newly formed trace, i.e. a trace cache miss or misprediction.) A person of ordinary skill in the art at the time the invention was made, and as taught by Rotenberg, would have recognized that the trace cache system provides fast trace-level sequencing while providing a method to create nonexistent traces or repair mispredicted traces (Rotenberg Section 2.1 Trace-Level Sequencing "…The trace predictor and trace cache together provide fast trace-level sequencing…Instruction-level sequencing, discussed in the next section, is required to construct nonexistent traces or repair trace mispredictions."). Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the trace caches, buffers, and selector of

Rotenberg in the device of Kyker to increase the speed of trace-level sequencing while using instruction-level sequencing to create new or correct mispredicted traces.

23. Referring to claim 16, Kyker in view of Rotenberg has taught the processing engine of claim 15, wherein the instruction segments are extended blocks (Kyker column 1, lines 30-45 "…the target address, the backward taken branch, and any micro-ops between the two form a loop…"; column 2, line 59 to column 3, line 33 "…Exemplary trace T1 includes a total of four micro-ops…The second exemplary trace T2 includes the same loop, L_H, L_1, L_2, but does not include any micro-op preceding the loop itself…"; Figure 1; and Figure 2 – In regards to Kyker, the loop contains a plurality of instructions, i.e. block of instructions, with four instructions and includes the extra micro-op preceding the loop itself.).

24. Referring to claim 17, Kyker in view of Rotenberg the processing engine of claim of 15, wherein the instruction segments are traces (Kyker Abstract "…a cache unit, which includes a data array that stores traces…In one exemplary method of unrolling loops, the processor or trace cache unrolls loops…"; column 2, line 59 to column 3, line 33 "…when the trace cache determines that a loop is present, the trace cache continues to build the trace by building additional iterations of the loop until the trace is a minimal length…In other words, the trace cache builds the loop repeatedly until the trace is, for example, over two trace lines long…"; Figure 1; and Figure 2).

25. Referring to claim 18, Kyker in view of Rotenberg has taught the processing engine of claim 15, wherein the instruction segments are basic blocks (Kyker column 1, lines 30-45 "…the target address, the backward taken branch, and any micro-ops between the two form a loop…"; column 2, line 59 to column 3, line 33 "…Exemplary trace T1 includes a total of four micro-

ops…The second exemplary trace T2 includes the same loop, L_H, L_1, L_2, but does not include any micro-op preceding the loop itself…"; Figure 1; and Figure 2 – In regards to Kyker, the loop contains a plurality of instructions, i.e. block of instructions, with three instructions and does not include the extra micro-op preceding the loop itself.).

26. Referring to claim 19, Kyker has not taught the processing engine of claim 15, wherein the instruction segment cache system further comprises a segment predictor provided in communication with the segment cache. Rotenberg has taught the processing engine of claim 15, wherein the instruction segment cache system further comprises a segment predictor provided in communication with the segment cache (Rotenberg Section 2.1 Trace-Level Sequencing "…A *next trace predictor*[14] treats traces as basic units and explicitly predicts sequences of traces…" and Figures 2). A person of ordinary skill in the art at the time the invention was made, and as taught by Rotenberg, would have recognized that the trace predictor achieves high branch prediction throughput with a single prediction per cycle (Rotenberg Section 2.1 Trace-Level Sequencing "…high branch prediction throughput is implicitly achieved with only a single trace prediction per cycle…"). Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the trace predictor of Rotenberg in the device of Kyker to achieve high branch prediction throughput in a single prediction cycle.

27. Examiner notes that the previous rejection of the claims under 35 U.S.C. §103 has been reintroduced. The rejection had been previously withdrawn due to the arguments presented in the Appeal Brief filed 20 December 2006 taken in light of the Federal Circuit Court's ruling on

Teleflex. However, the subsequent ruling from the U.S. Supreme Court on KSR has forced the Examiner to re-evaluate that interpretation of the arguments and prior art rejections.

28. Claims 1, 3-8, 12-15 and 17-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Patel et al., *Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing*, in further view of Johnson, U.S. Patent No. 5,924,092.

29. Regarding claim 1, Patel has taught a cache comprising a cache line to store an instruction segment further comprising a plurality of instructions stored in sequential positions of the cache line (see Col.1 line 26 – Col.2 line 15). Patel has not explicitly taught storing the plurality of instructions in sequential positions of a cache line in reverse program order.

30. However, Johnson has taught the storing of blocks of data in reverse order so that those blocks that were in the first block of a data structure that are frequently accessed and modified will require less moving and fewer modifications after being placed at the last location of data structure, the fewer modifications resulting in improved performance (see Johnson, Col.4 lines 13-24). Because the traces of Patel are indexed, as well as accessed and modified, via their head (first) instructions (see Patel, Col.4 lines 23-31 and Col.5 lines 12-18), one of ordinary skill in the art would have found it obvious to modify the instruction segment of Patel to store the instructions of the instruction trace in reverse order so that the frequently accessed and modified head of the trace will be moved and modified fewer times so that performance is improved.

31. Regarding claim 3, Patel in view of Johnson has taught the cache of claim 1, wherein the instruction segment is a trace (see Patel, Col.3 lines 2-12).

32. Regarding claim 4, Patel in view of Johnson has taught the cache of claim 1, wherein the instruction segment is a basic block (see Patel, Col.2 lines 2-5).

33. Regarding claim 5, Patel has taught a segment cache (see "trace cache" of Fig.1) for a front-end system in a processor, comprising a plurality of cache entries to store instruction segments (see Col.1 line 26 – Col.2 line 15). Patel has not explicitly taught storing the instruction segments in reverse program order.

34. However, Johnson has taught the storing of blocks of data in reverse order so that those blocks that were in the first block of a data structure that are frequently accessed and modified will require less moving and fewer modifications after being placed at the last location of data structure, the fewer modifications resulting in improved performance (see Johnson, Col.4 lines 13-24). Because the traces of Patel are indexed, as well as accessed and modified, via their head (first) instructions (see Patel, Col.4 lines 23-31 and Col.5 lines 12-18), one of ordinary skill in the art would have found it obvious to modify the instruction segment of Patel to store the instructions of the instruction trace in reverse order so that the frequently accessed and modified head of the trace will be moved and modified fewer times so that performance is improved.

35. Regarding claim 6, Patel in view of Johnson has taught an apparatus comprising:

a. An instruction cache system (see Patel, "instruction cache" of Fig.1),

b. An instruction segment system, comprising:

i. A fill unit (see Patel, "fill unit" of Fig.1) provided in communication with the instruction cache system, the segment cache of claim 5 included therein (see Patel, Fig.1),

c. A selector (see Patel, "selection logic" of Fig.1) coupled to an output of the instruction cache system and to an output of the segment cache (see Patel, Fig.1).

36. Regarding claim 7, Patel in view of Johnson has taught the front-end system of claim 6, wherein the instruction segment system further comprises a segment predictor (see Patel, "multiple branch predictor" of Fig.1) provided in communication with the segment cache. Here, when the multiple branch predictor is coupled with the trace cache and mediated by the selection logic, it effectively predicts segments because the basic blocks stored in the trace cache all begin with branches (see Patel, Col.5 lines 5-29).

37. Regarding claim 8, Patel has taught a method for storing instruction segments in a processor, comprising:

a. Building an instruction segment based on program flow (see Col.1 line 26 – Col.2 line 15),

b. Storing the instruction segment in a cache (see Col.1 line 26 – Col.2 line 15).

38. Patel has not explicitly taught wherein the instruction segment is stored in reverse program order.

39. However, Johnson has taught the storing of blocks of data in reverse order so that those blocks that were in the first block of a data structure that are frequently accessed and modified will require less moving and fewer modifications after being placed at the last location of data structure, the fewer modifications resulting in improved performance (see Johnson, Col.4 lines 13-24). Because the traces of Patel are indexed, as well as accessed and modified, via their head (first) instructions (see Patel, Col.4 lines 23-31 and Col.5 lines 12-18), one of ordinary skill in the art would have found it obvious to modify the instruction segment of Patel to store the instructions of the instruction trace in reverse order so that the frequently accessed and modified head of the trace will be moved and modified fewer times so that performance is improved.

40. Regarding claim 12, Patel in view of Johnson has taught the method of claim 8, wherein the instruction segment is a trace (see Patel, Col.3 lines 2-12).

41. Regarding claim 13, Patel in view of Johnson has taught the method of claim 8, wherein the instruction segment is a basic block (see Patel, Col.2 lines 2-5).

42. Regarding claim 14, Patel has taught a processing engine, comprising:

a. A front-end stage to build and store instruction segments (see Col.1 line 26 – Col.2 line 15),

b. An execution unit (see "HPS Execution Core" in Fig.1) in communication with the front end stage (see Col.5 line 30 – Col.6 line 4).

43. Patel has not explicitly taught building and storing the instruction segments in reverse program order.

44. However, Johnson has taught the storing of blocks of data in reverse order so that those blocks that were in the first block of a data structure that are frequently accessed and modified will require less moving and fewer modifications after being placed at the last location of data structure, the fewer modifications resulting in improved performance (see Johnson, Col.4 lines 13-24). Because the traces of Patel are indexed, as well as accessed and modified, via their head (first) instructions (see Patel, Col.4 lines 23-31 and Col.5 lines 12-18), one of ordinary skill in the art would have found it obvious to modify the instruction segment of Patel to store the instructions of the instruction trace in reverse order so that the frequently accessed and modified head of the trace will be moved and modified fewer times so that performance is improved.

45. Regarding claim 15, Patel in view of Johnson has taught the processing engine of claim 14, wherein the front-end stage comprises:

a. An instruction cache system (see Patel, "instruction cache" of Fig.1),

b. An instruction segment system, comprising:

i. A fill unit (see Patel, "fill unit" of Fig.1) provided in communication with the instruction cache system (see Patel, Fig.1),

ii. A segment cache (see "trace cache" of Fig.1),

c. A selector (see Patel, "selection logic" of Fig.1) coupled to an output of the instruction cache system and to an output of the segment cache (see Patel, Fig.1).

46. Regarding claim 17, Patel in view of Johnson has taught the method of claim 15, wherein the instruction segments are traces (see Patel, Col.3 lines 2-12).

47. Regarding claim 18, Patel in view of Johnson has taught the method of claim 15, wherein the instruction segments are basic blocks (see Patel, Col.2 lines 2-5).

48. Regarding claim 19, Patel in view of Johnson has taught the method of claim 15, wherein the instruction segment cache system further comprises a segment predictor (see Patel, "multiple branch predictor of Fig.1) provided in communication with the segment cache. Here, when the multiple branch predictor is coupled with the trace cache and mediated by the selection logic, it effectively predicts segments because the basic blocks stored in the trace cache all begin with branches (see Patel, Col.5 lines 5-29).

49. Claims 2, 9-11 and 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Patel et al., *Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing*, in view of Johnson, U.S. Patent No. 5,924,092, in further view of Peled et al., U.S. Patent No. 6,076,144.

50. Regarding claim 2, Patel in view of Johnson has taught the cache of claim 1, but has not explicitly taught wherein the instruction segment is an extended block.

51. However, Peled has taught trace segments which have multiple entry points and a single exit that allow redundant code segments to be eliminated from the trace cache, thereby improving cache utilization (see Peled, Col.1 lines 60-63, Col.4 lines 13-37, and Fig.3). Because the specification has defined an extended block to have multiple entry points and a single exit point (see p.2 of Specification), one of ordinary skill in the art would have found it obvious to modify the instruction segments of Patel to allow for multiple entry points and a single exit so that redundant code segments could be eliminated from the trace cache and performance could be improved.

52. Regarding claim 9, Patel in view of Johnson have taught the method of claim 9, but have not explicitly taught wherein the method further comprises:

a. Building a second instruction segment based on program flow,

b. If the first and second instruction segments overlap, extending the first instruction segment to include non-overlapping instructions from the second instruction segment.

53. However, Peled has taught building a second instruction segment based on program flow and subsequently extending the first instruction segment to include the non-overlapping instructions from the second instruction segment if the two segments overlap (see Col.4 lines 13-37) in order to reduce the degree of code redundancy in the trace cache (see Col.1 lines 60-63). One of ordinary skill in the art would have recognized that it is desirable to reduce redundancy within a trace cache so that the cache can be more effectively used and more different traces

stored. Therefore, one of ordinary skill in the art would have found it obvious to extend an existing instruction segment to include non-overlapping instructions from a second instruction segment in order to reduce trace cache redundancy.

54. Regarding claim 10, Patel in view of Johnson in further view of Peled has taught the method of claim 9, but has not explicitly taught wherein the extending comprises storing the non-overlapping instructions in the cache in reverse program order in successive cache positions adjacent to the instructions from the first instruction segment.

55. However, Patel in view of Johnson has taught that instructions in instruction segments are stored in reverse program order (see paragraphs 21-23 above). Because, an extended segment is still an instruction segment, one of ordinary skill in the art would have found it obvious to also store the extended instruction segments in reverse program order.

56. Regarding claim 11, Patel in view of Johnson has taught the method of claim 8, but has not explicitly taught wherein the instruction segment is an extended block.

57. However, Peled has taught trace segments which have multiple entry points and a single exit that allow redundant code segments to be eliminated from the trace cache, thereby improving cache utilization (see Peled, Col.1 lines 60-63, Col.4 lines 13-37, and Fig.3). Because the specification has defined an extended block to have multiple entry points and a single exit point (see p.2 of Specification), one of ordinary skill in the art would have found it obvious to modify the instruction segments of Patel to allow for multiple entry points and a single exit so that redundant code segments could be eliminated from the trace cache and performance could be improved.

58. Regarding claim 16, Patel in view of Johnson has taught the method of claim 15, but has not explicitly taught wherein the instruction segments are extended blocks.

59. However, Peled has taught trace segments which have multiple entry points and a single exit that allow redundant code segments to be eliminated from the trace cache, thereby improving cache utilization (see Peled, Col.1 lines 60-63, Col.4 lines 13-37, and Fig.3). Because the specification has defined an extended block to have multiple entry points and a single exit point (see p.2 of Specification), one of ordinary skill in the art would have found it obvious to modify the instruction segments of Patel to allow for multiple entry points and a single exit so that redundant code segments could be eliminated from the trace cache and performance could be improved.

Response to Arguments

60. Examiner withdraws the 101 rejections in favor of the arguments.

61. Examiner withdraws the 112 rejections in favor of the arguments.

62. Applicant's arguments filed 03 March 2008, with regards to the prior art rejections, have been fully considered but they are not persuasive.

63. Applicant argues in essence on pages 7-10,

> Applicants submit the cited section does not teach or suggest a cache comprising a cache line to store an instruction segment further comprising a plurality of instructions stored in sequential positions of the cache line in reverse program order...

64. This has not been found persuasive. Applicant sites page 5, line 21 to page 6, line 12 as the "clarification" on the meaning of "reverse program order". However, this is not a clear,

definite, deliberate definition of the phrase “reverse program order”, nor has the Examiner been able to locate a definition in any dictionary or paper to support arguments that "reverse program order" has the specific, clear, and narrow definition Applicants' arguments suggest. As cited, the claim limitation in question is storing an instruction segment in “reverse program order”. As explained in the previous rejection and copied above, Kyker shows in Figure 2 and explains the in the cited lines, the loop has three main instructions and, when the backwards branch is encountered, the program jumps backwards, i.e. reverses directions, to the beginning of the loop instead of moving forward in the program. As shown in Figure 2, when L_{11} is encountered, previously stored instructions L_2 and L_3 are stored again in consecutive lines in the cache, e.g. the trace cache reverses the program direction and stores previously stored instructions again in the trace cache when a loop is encountered. Applicant’s arguments appear to suggest that the phrase “reverse program order” has a specific, accepted meaning in the art, but the Examiner could not locate this specific meaning within the specification nor in the art. As such, the Examiner read the phrase “reverse program order” to mean the program stored in the cache line is in the opposite direction of the normal program direction. Kyker meets this limitations since they store in the trace cache instructions previously stored in the trace cache when the program direction is reversed by a backwards branch from a loop. The program is no longer stored in the trace cache in normal program order, i.e. in a forward program direction, because the backwards loop requires the program to branch to an instruction previously stored in the trace cache, thereby forcing the reverse program order by making the program direction go backwards to re-execute instructions in the loop.

65. Applicants’ argue in essence on pages 10-14

> …Patel neither teaches no suggests that the fetch rates may be improved by reversing the order of the instructions in the traces…

66. This has not been found persuasive. The rejection did not rely upon Patel to teach reversing the order of the instructions in the cache. Johnson was relied upon. Please see the rejection above.

67. Applicants' argue in essence on pages 13-14

> …This cited section does not, however, discuss improving overall performance during *accessing* or *using* data in the data array. *Accessing* data or *using* data is not the same as *modifying* data, and the cited section of Johnson does not discuss the benefits of its sorting algorithm during data access or data "use" at all...

68. This has not been found persuasive. To modify something, such as data, means that the data is changed in some manner. Data cannot be changed without it being accessed or making use of the data. In order to modify a piece of data or information, the original data must be accessed or used for the change to occur. To argue that "modifying" data does not mean that the data is "accessed" or "used" is illogical and is contrary to the understanding of a person of ordinary skill in the art, since no change or modification to data can take place without using the data.

Conclusion

69. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

70. A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO

MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

71. Any inquiry concerning this communication or earlier communications from the examiner should be directed to AIMEE J. LI whose telephone number is (571)272-4169. The examiner can normally be reached on M-T 7:00am-4:30pm.

72. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

73. Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Aimee J Li/
Primary Examiner, Art Unit 2183
9 June 2008